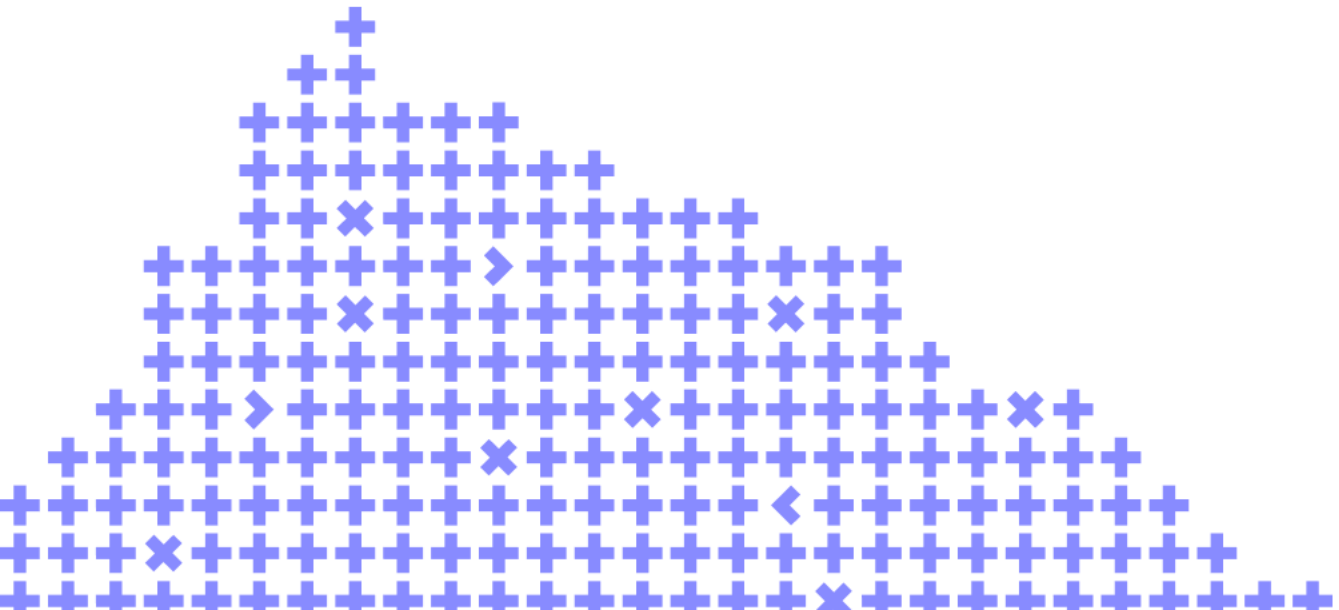
Level up your optimization process: how to implement distributed profiling and why you want to have it

Igor Solovyov



High Load Armenia

Co-organizer

Yandex

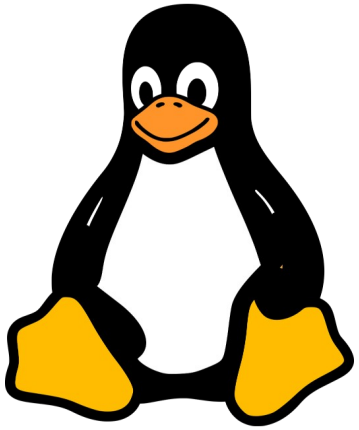
About me

- Developer in the Yandex infrastructure advertising team
- Develop a system with one million requests per second
- Specialize in data structure optimization and big data analysis
- Develop a distributed profiling system

Contact me
in Telegram:

Definitions

GNU Gprof

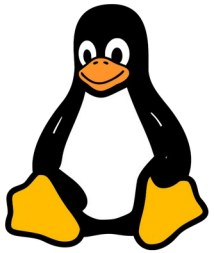


perf

Profiling

- Profiler is a performance analysis tool for applications

GNU Gprof

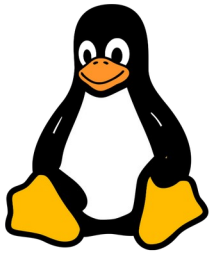


perf

Distributed profiling

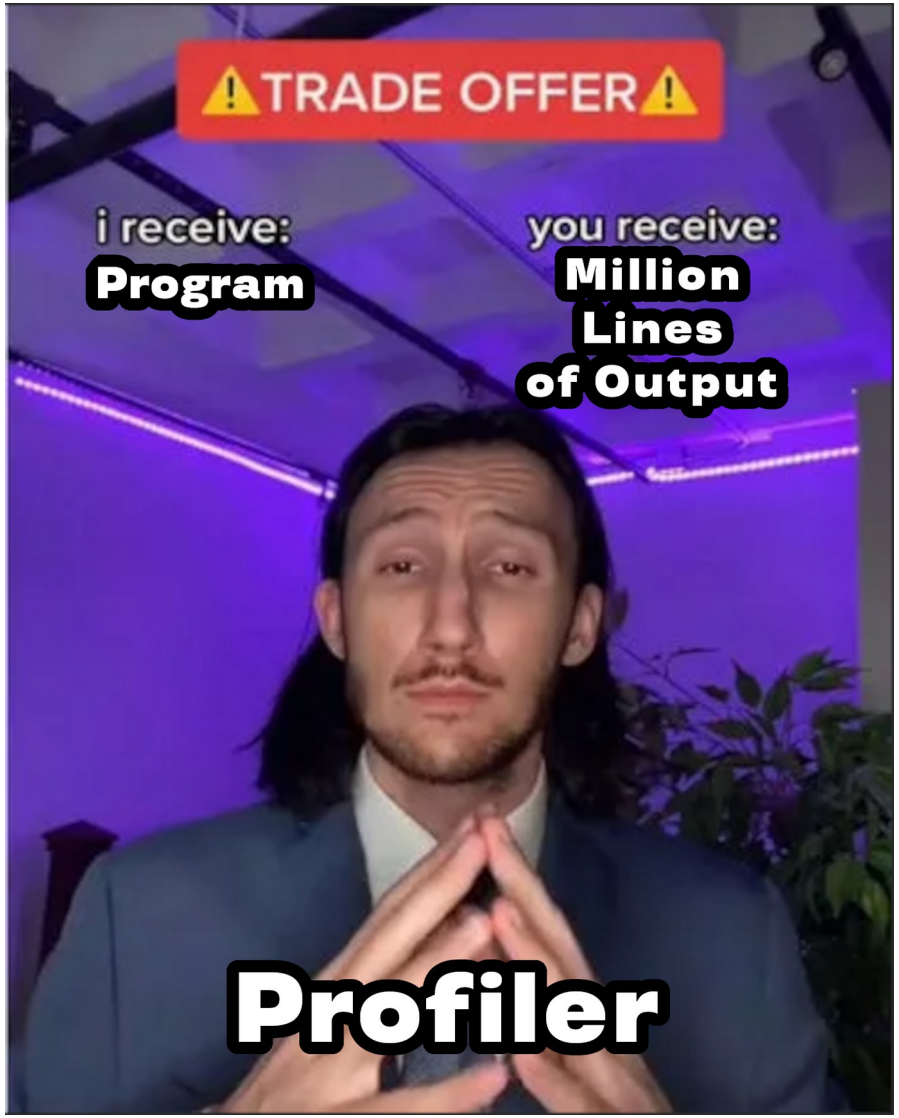
- Distributed profiler is a performance analysis tool for distributed applications that aggregates data from multiple hosts

GNU Gprof

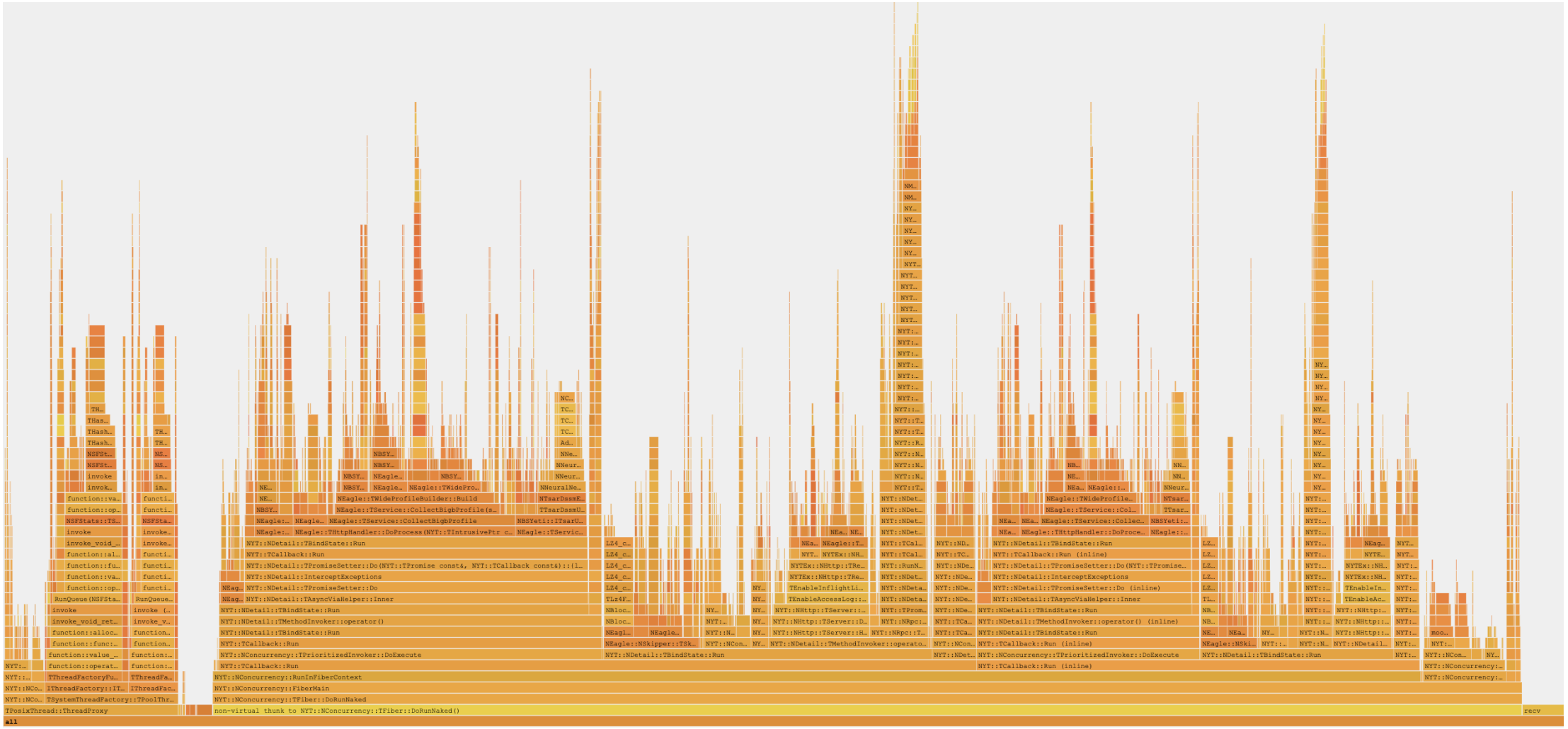


perf

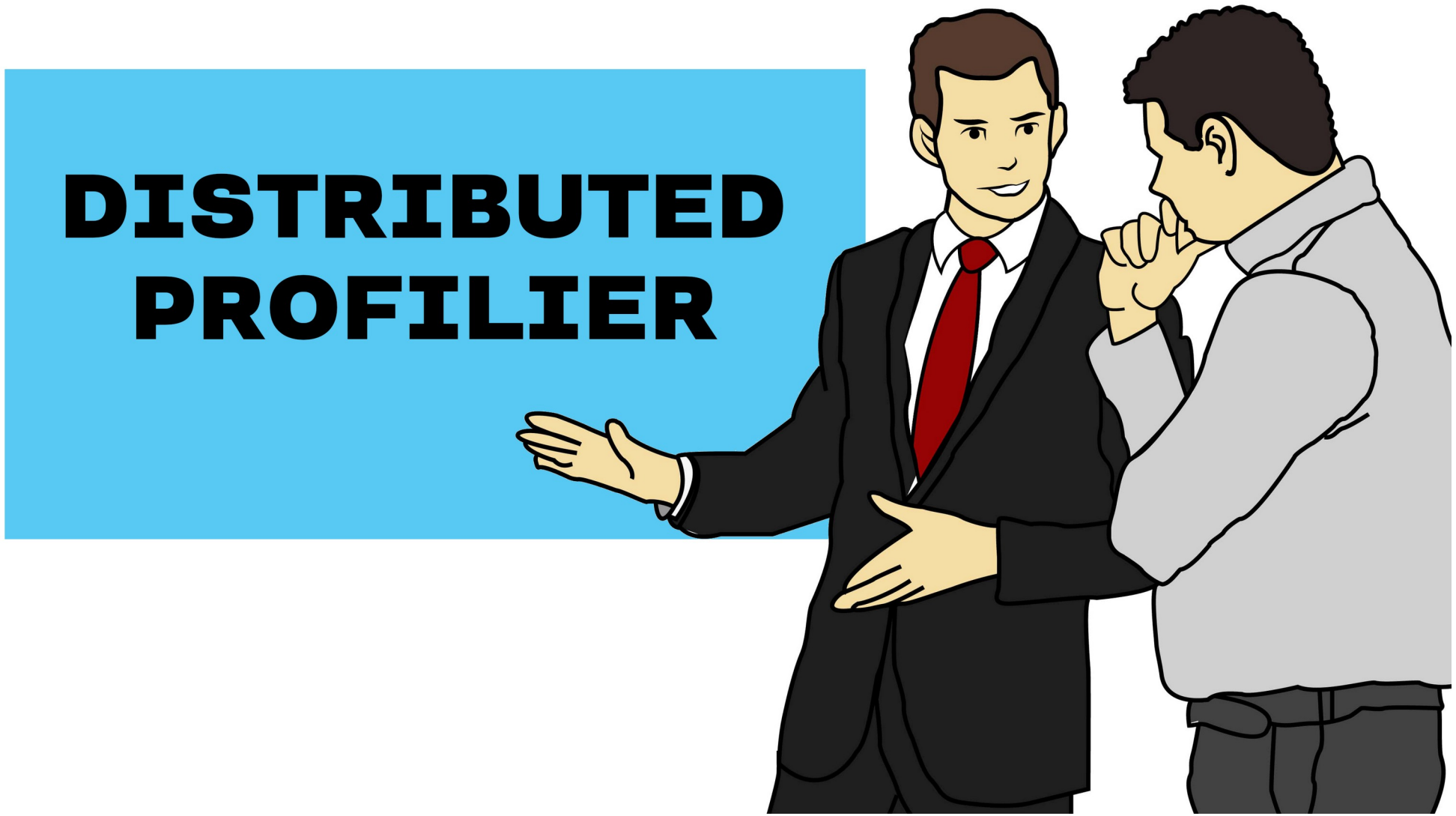
Profiler


⚠TRADE OFFER⚠
i receive:
Program
you receive:
Million
Lines
of Output
Profiler

Flame graphs



Motivation



1. Exploration of targets for optimization

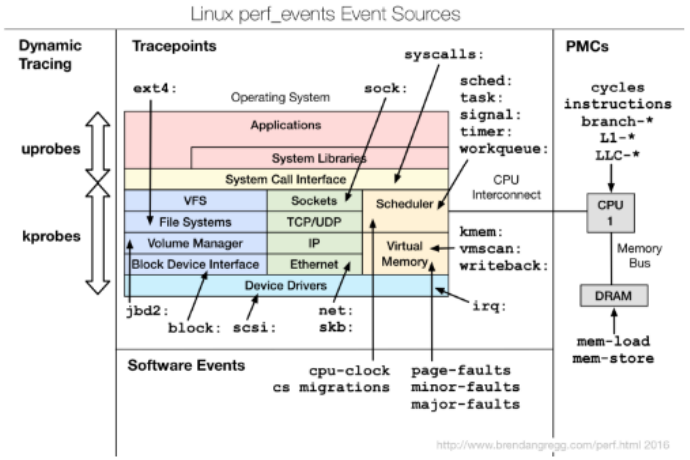
2. Complexity of using conventional local profilers

perf Examples

These are some examples of using the perf Linux profiler, which has also been called Performance Counters for Linux (PCL), Linux perf events (LPE), or perf_events. Like Vince Weaver, I'll call it perf_events so that you can search on that term later. Searching for just "perf" finds sites on the police, petroleum, weed control, and a T-shirt. This is not an official perf page, for either perf_events or the T-shirt.

perf_events is an event-oriented observability tool, which can help you solve advanced performance and troubleshooting functions. Questions that can be answered include:

- Why is the kernel on-CPU so much? What code-paths?
- Which code-paths are causing CPU level 2 cache misses?
- Are the CPUs stalled on memory I/O?
- Which code-paths are allocating memory, and how much?
- What is triggering TCP retransmits?
- Is a certain kernel function being called, and how often?
- What reasons are threads leaving the CPU?

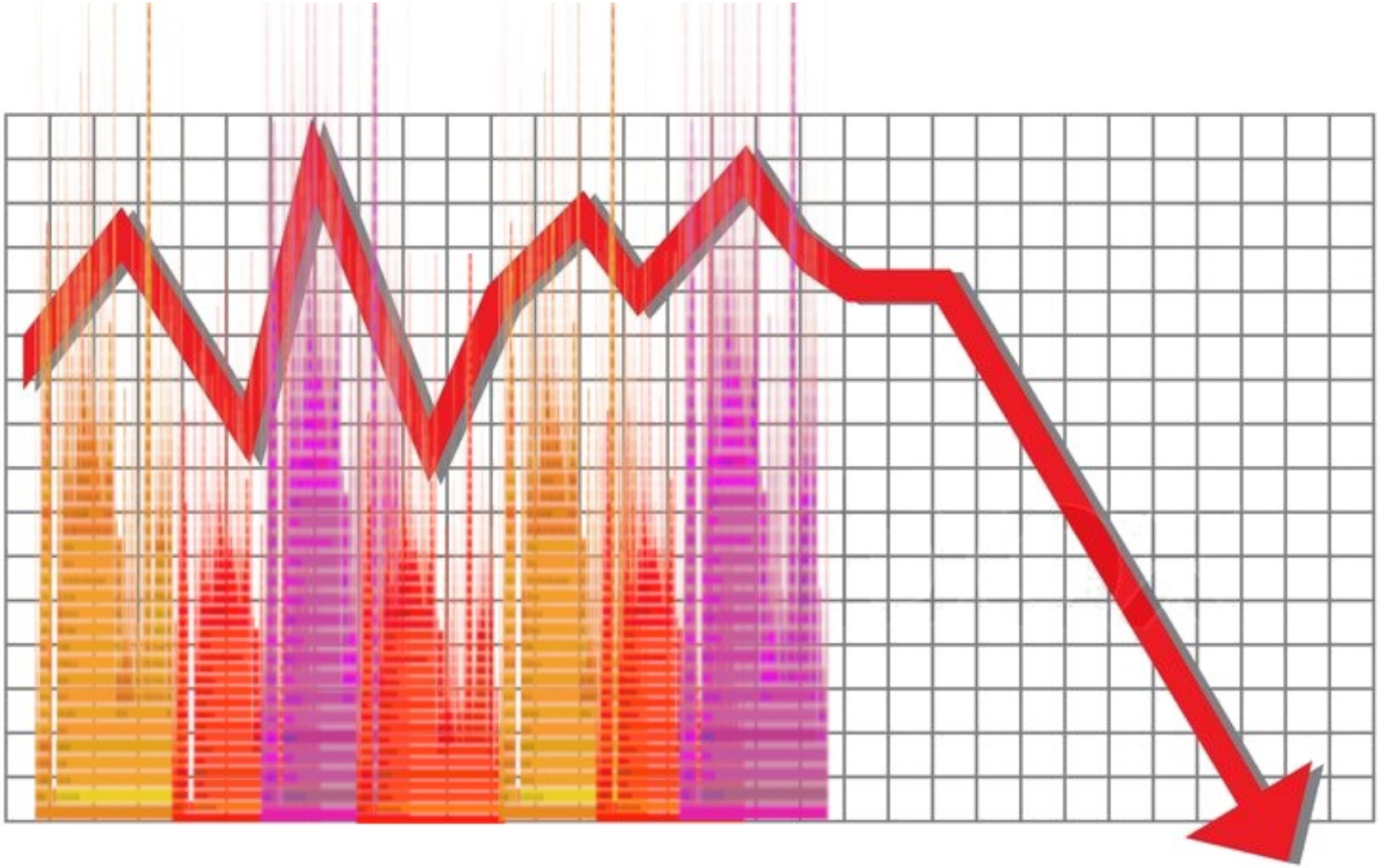


perf_events is part of the Linux kernel, under tools/perf. While it uses many Linux tracing features, some are not yet exposed via the perf command, and need to be used via the ftrace interface instead. My perf-tools collection (github) uses both perf_events and ftrace as needed.

This page includes my examples of perf_events. A table of contents:

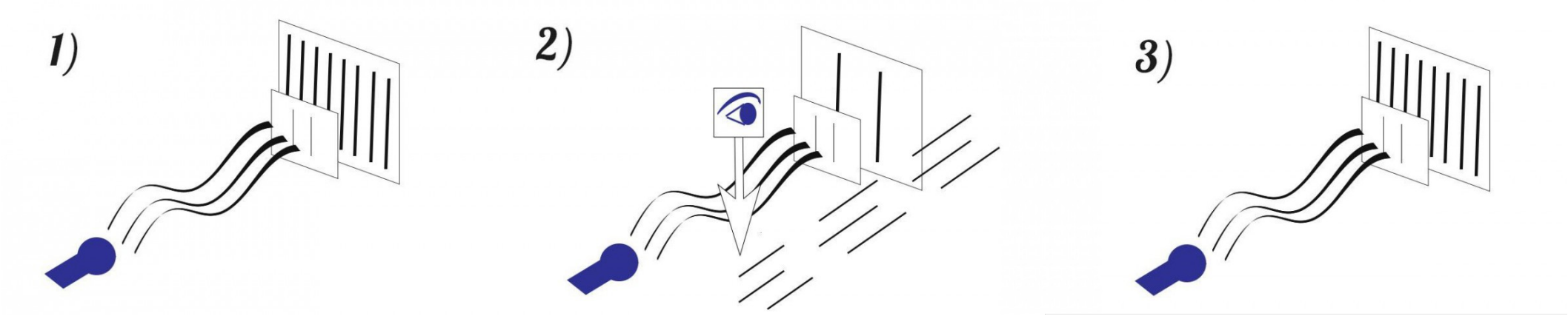
3. Waste of time on every request

4. Difficulty of a historical comparison



5. Impact on the performance of the application being profiled

- You start to profile a host
- There is only one host, so sampling is frequent
- The host degrades in comparison with other hosts
- Balancers try to ignore the host
- You have measured a degraded host with an unrepresentative workload



Theory

Sampling and instrumentation profilers

Sampling profiler

- perf
- OProfile

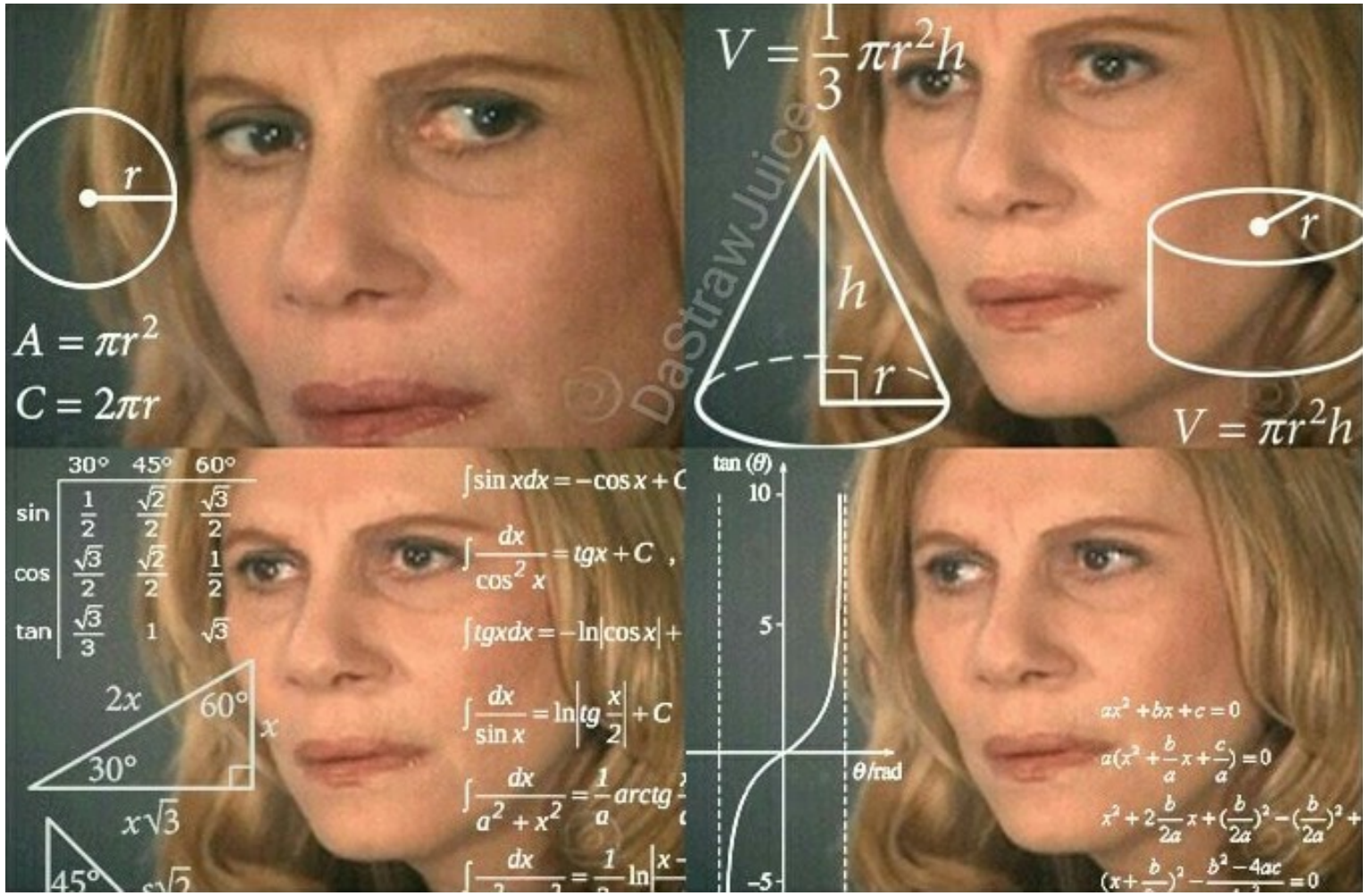
Instrumentation profiler

- Manual
- Automatic source level
- Intermediate language
- Compiler assisted
- Binary translation
- Runtime instrumentation
- Runtime injection

Instrumentation profilers' limitations

- Performance changes: stack trace writing is too expensive
- Heisenbugs: writing a stack trace may change the execution order

Sampling profiler math



Sampling profiler math: definitions

1. The profiler pauses the program at a random point and prints a stack trace.
2. $X :=$ printed stack.
3. The sample space of the random variable X is the set of all possible stacks:

$$\mathbf{S} = \{S \mid S \textit{ is a stack in program}\}$$

Definition

Indicator random variable for a stack S:

$$I_S(X) = \begin{cases} 1 & \text{if } x = S\ , \\ 0 & \text{if } x \neq S\ . \end{cases}$$

Sampling profiler math: asymptotic distribution

1. Consider the following sequence:

$$I_S^N := \frac{\sum_{i=1}^{N} (I(X_i) - \mu)}{\sqrt{N}}$$

2. According to the C.L.T:

$$\lim_{N \to \infty} I_S^N \to \mathcal{N}(0, \sigma^2)$$

3. Asymptotically:

$$\overline{I_S^N} = \frac{1}{N} \sum_{i=1}^{N} I(X_i) = \frac{I_S^N}{\sqrt{N}} + \mu \approx \mathcal{N}(\mu, \sigma^2/N)$$

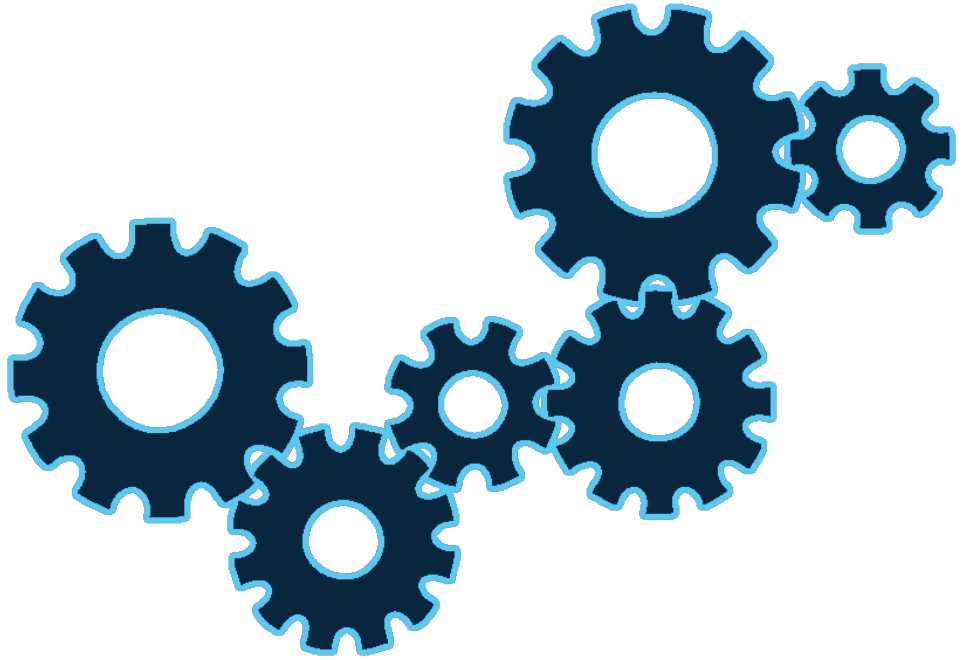
Sampling profiler math: relative error and examples

1. Relative error:
$$e = X(N)/\mu - 1 \approx \mathcal{N}(\mu, \frac{1}{\mu N})$$
2. According to the three-sigma rule:
$$N \sim e^2/\mu$$
3. Particular solutions:

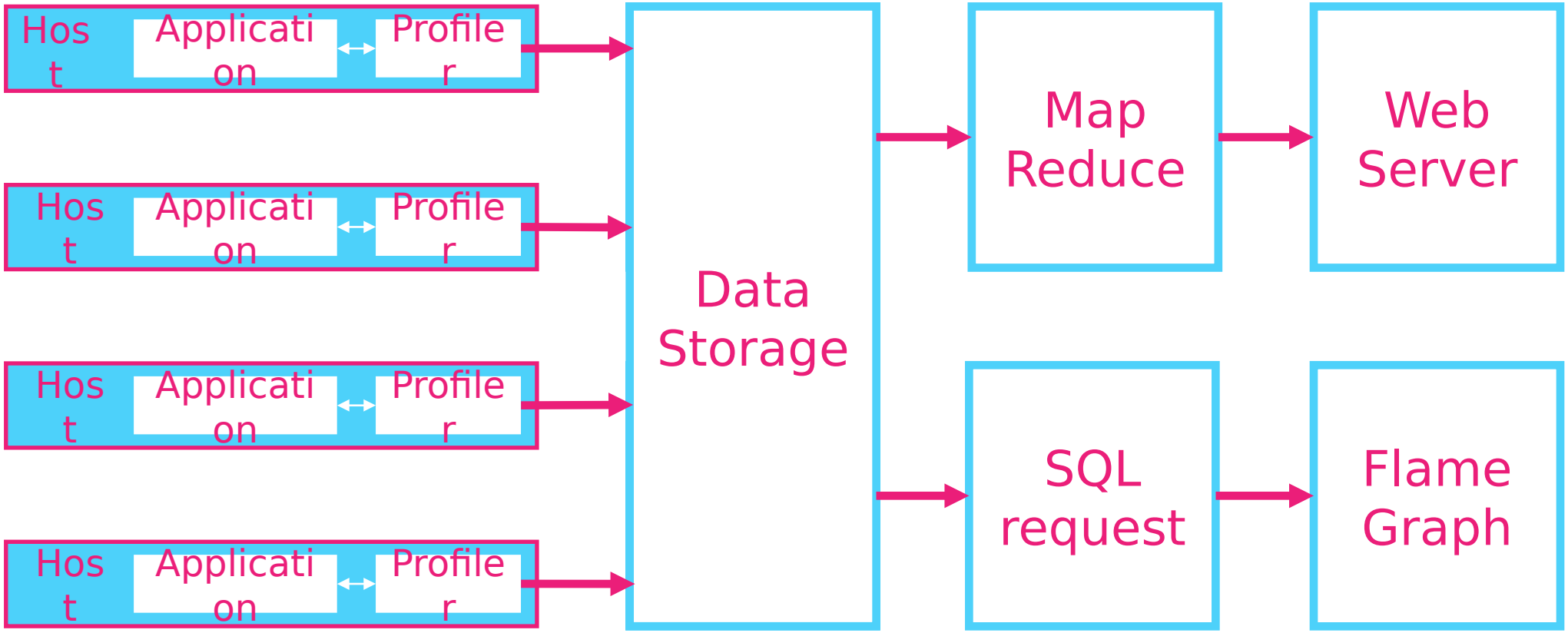
Examples

1. $$e = 0.1, \mu = 0.01 \implies N = 10'000 \ (3\sigma\text{-case} : 90'000)$$
2. $$e = 0.1, \mu = 0.001 \implies N = 100'000 \ (3\sigma\text{-case} : 900'000)$$

How it works



General scheme



Stack extractors: poor mans profiler

For more information
on Poor Mans Profiler

Stack extractors: poor mans profiler

```
gdb.cmd:
    set pagination 0
    define print_
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack extractors: poor mans profiler

```
gdb.cmd:
    set pagination 0
    define print_
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack extractors: poor mans profiler

```
gdb.cmd:
    set pagination 0
    define print_
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack extractors: poor mans profiler

```
gdb.cmd:
    set pagination 0
    define print_
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack extractors: poor mans profiler

```
gdb.cmd:
    set pagination 0
    define print_
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack extractors: poor mans profiler

```
gdb.cmd:
    set pagination 0
    define print_
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack extractors: poor mans profiler

```
gdb.cmd:
    set pagination 0
    define print_
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack extractors: signal-based profilers

Conference talk
«Query Profiler: The Difficult Path»
(in Russian)

Signal-based profiler library

Stack marking

Service	Stack	Timestamp	Experiments	Count
worker	all DoRunNaked() FiberMain RunInFiberContext TPrioritizedInvoker::DoExecute THttpHandler::Handle**Critical**Request() CalculateMillionthFibbonacciNumberRecursively() CalculateMillionthFibbonacciNumberRecursively() ...	1500000000	**AddSpecialFeatureStoreLessData**	1

Stack marking: Code

```
namespace {
    class RequestInfoKeeper {
        static thread_local std::string Request;
    public:
        static void SetRequest(const std::string& request) {
            Request = request;
        }

        static const std::string& GetRequest() {
            return Request;
        }
    };

    thread_local std::string RequestInfoKeeper::Request("default");
};

const std::string& GetRequest() {
    return RequestInfoKeeper::GetRequest();
}

void SetRequest(const std::string& request) {
    return RequestInfoKeeper::SetRequest(request);
}
```

Stack marking: Code

```
namespace {
    class RequestInfoKeeper {
        static thread_local std::string Request;
    public:
        static void SetRequest(const std::string& request) {
            Request = request;
        }

        static const std::string& GetRequest() {
            return Request;
        }
    };

    thread_local std::string RequestInfoKeeper::Request("default");
};

const std::string& GetRequest() {
    return RequestInfoKeeper::GetRequest();
}

void SetRequest(const std::string& request) {
    return RequestInfoKeeper::SetRequest(request);
}
```

Stack marking: Code

```
namespace {
    class RequestInfoKeeper {
        static thread_local std::string Request;
    public:
        static void SetRequest(const std::string& request) {
            Request = request;
        }

        static const std::string& GetRequest() {
            return Request;
        }
    };

    thread_local std::string RequestInfoKeeper::Request("default");
};

const std::string& GetRequest() {
    return RequestInfoKeeper::GetRequest();
}

void SetRequest(const std::string& request) {
    return RequestInfoKeeper::SetRequest(request);
}
```

Stack marking: Code

```
namespace {
    class RequestInfoKeeper {
        static thread_local std::string Request;
    public:
        static void SetRequest(const std::string& request) {
            Request = request;
        }

        static const std::string& GetRequest() {
            return Request;
        }
    };

    thread_local std::string RequestInfoKeeper::Request("default");
};

const std::string& GetRequest() {
    return RequestInfoKeeper::GetRequest();
}

void SetRequest(const std::string& request) {
    return RequestInfoKeeper::SetRequest(request);
}
```

Stack marking: Gdb

```
gdb.cmd:
    set pagination 0
    define print_
        print GetRequest()
        bt
    end
    thread apply all print_
    quit

bash:
    gdb --quiet --batch -x gdb.cmd --pid $PID > result.txt
```

Stack aggregation

- Aggregation by a day
- Partial sums

Stack aggregation

- Aggregation by a day
- Partial sums

Service	Stack	Timestamp	Experiments	Count
worker	all DoRunNaked() FiberMain RunInFiberContext TPrioritizedInvoker::DoExecute THttpHandler::Handle**Critical**Request() CalculateMillionthFibbonacciNumberRecursively() CalculateMillionthFibbonacciNumberRecursively() ...	1500000000	AddSpecialFeature StoreLessData	1

Stack aggregation

- Aggregation by a day
- Partial sums

Service	Stack	Timestamp	Experiments	Count
worker	**allDoRunNaked()FiberMain RunInFiberContext TPrioritizedInvoker::DoExecute THttpHandler::HandleCriticalRequest() CalculateMillionthFibbonacciNumberRecursively() CalculateMillionthFibbonacciNumberRecursively()...**	1500000000	AddSpecialFeature StoreLessData	1

Stack aggregation

- Aggregation by a day
- Partial sums

Service	StackId	Timestamp	Experiments	Count
worker	**4815162342**	1500000000	AddSpecialFeature StoreLessData	1

Stack aggregation

- Aggregation by a day
- Partial sums

Service	StackId	Timestamp	Experiments	Count
worker	4815162342	**1500000000**	AddSpecialFeature StoreLessData	1

Stack aggregation

- Aggregation by a day
- Partial sums

Service	StackId	Date	Experiments	Count
worker	4815162342	**Fri Jul 14 2017**	AddSpecialFeature StoreLessData	1

Stack aggregation

- Aggregation by a day
- Partial sums

Service	StackId	Date	Experiments	Count
worker	4815162342	Fri Jul 14 2017	**AddSpecialFeatureStoreLessData**	1

Stack aggregation

- Aggregation by a day
- Partial sums

Service	StackId	Date	Experiments	Count
worker	4815162342	Fri Jul 14 2017	**Default**	1
worker	4815162342	Fri Jul 14 2017	**AddSpecialFeature**	1
worker	4815162342	Fri Jul 14 2017	**StoreLessData**	1

Stack aggregation

- Aggregation by a day
- Partial sums

Service	StackId	Date	Experiments	Count
worker	4815162342	Fri Jul 14 2017	AddSpecialFeature	**1**
worker	4815162342	Fri Jul 14 2017	AddSpecialFeature	**1**
worker	4815162342	Fri Jul 14 2017	AddSpecialFeature	**1**

Stack aggregation

- Aggregation by a day
- Partial sums

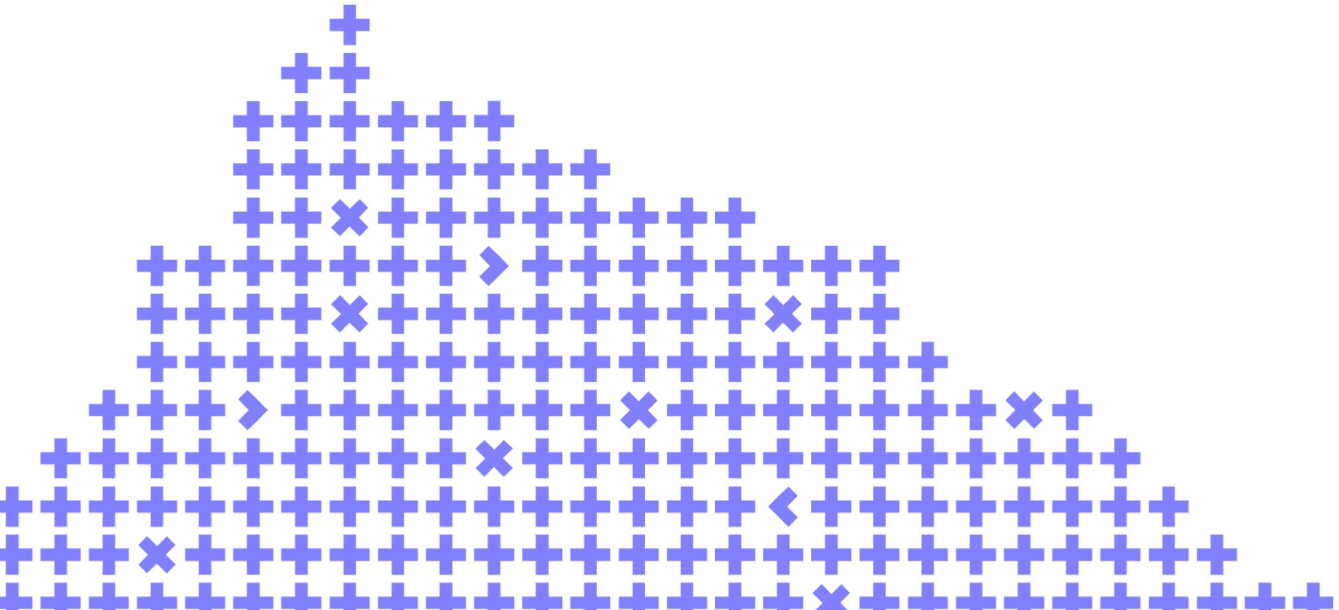
Service	StackId	Date	Experiments	Count
worker	4815162342	Fri Jul 14 2017	AddSpecialFeature	3

Conclusion

- Profiling is important
- Profiling is complicated
- Distributed profiling may be a solution

Leave your feedback!

You can rate the talk and give a feedback on what you've liked or what could be improved



High Load++ Armenia

Co-organizer

Yandex